The R language

Martin Frigaard

2021-08-27

Contents

List of Tables

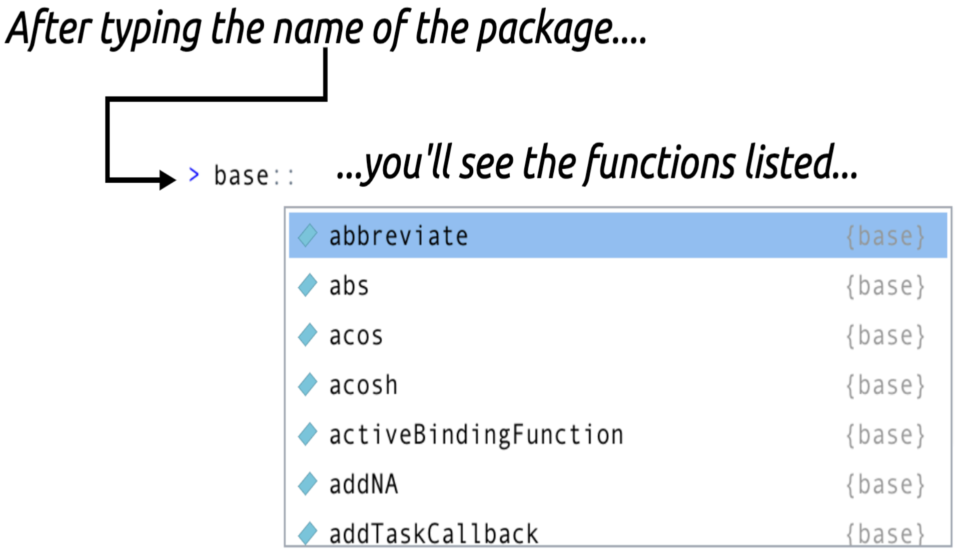
List of Figures

R is an object-oriented, functional programming language for statistical analysis and graphics. R is also a free and open-source software (FOSS) with a massive global community of users and developers who have helped create and maintain tools for data manipulation, graphics, statistics, and machine learning.

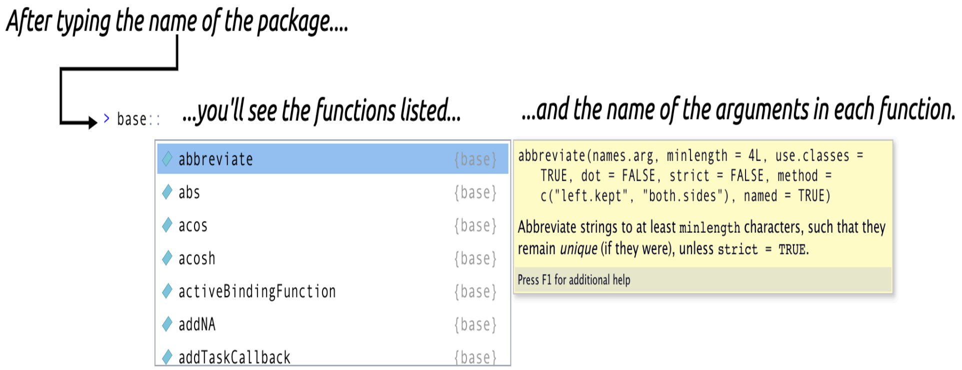
Base R packages

R `packages` are collections of commands for a particular purpose or task. R comes 'out of the box' with a ton of functions for manipulating, analyzing, and visualizing data. Two of the most commonly used standard packages are `base` and `utils`.

You can access any function in a package using the `package::function()` syntax. If you're in RStudio, you can actually see the functions in each package by using the tab-completion feature:


After typing the name of the package....
> base::
...you'll see the functions listed...
abbreviate {base}
abs {base}
acos {base}
acosh {base}
activeBindingFunction {base}
addNA {base}
addTaskCallback {base}

If you hover over the function with your mouse cursor, you'll also see the arguments and documentation for each function.


After typing the name of the package....
> base::
...you'll see the functions listed...
...and the name of the arguments in each function.
abbreviate {base}
abs {base}
acos {base}
acosh {base}
activeBindingFunction {base}
addNA {base}
addTaskCallback {base}
Press F1 for additional help

```
abbreviate(names.arg, minlength = 4L, use.classes =
    TRUE, dot = FALSE, strict = FALSE, method =
    c("left.kept", "both.sides"), named = TRUE)
```

Abbreviate strings to at least `minlength` characters, such that they remain *unique* (if they were), unless `strict = TRUE`.

When you're using the `utils::install.packages()` function, the package files are installed from the Comprehensive R Archive Network, or CRAN. These packages have passed a variety of tests and are generally considered to be more reliable.

User-written packages

Most of the packages we'll be using in this course come from the `tidyverse`, which is a suite of tools pioneered by RStudio's Chief Scientist Hadley Wickham. All packages in the `tidyverse` work well together because they center around a common thread of tidy data.

To install and load the `tidyverse`, we will use the `utils::install.packages()` function to download and installs R packages into a local folder on our computer, and the `base::library()` command loads the packages.

```
install.packages("tidyverse")
library(tidyverse)
```

NOTE: Not all functions return an output. Some functions return messages (or prompts), so be sure to check the help files by using `?install.packages` in the console.

User-written packages can be installed from code repositories like Github.. The R ecosystem has over 10,000 user-written packages available on CRAN.

First you will need to install the `remotes` package from CRAN

```
install.packages("remotes")
```

Second, you load the package with `library()`

```
library(remotes)
```

Finally, we use the `remotes::install_github()` to download and install the `tidyverse` package.

```
remotes::install_github("tidyverse/tidyverse")
library(tidyverse)
```

Note: when installing packages from Github or other repos, you're getting the 'freshest' version, so there might be bugs or errors. If you run into an issue, look for a version of the package on CRAN

Functions and objects

The R language is comprised of functions and objects. R uses functions to perform operations (like `mean()`, `sum()`, `lm()` (for linear model)) on objects (vectors, arrays, matrices, data.frames or lists).

Generally speaking, functions are similar to *verbs*, and objects are more like *nouns*. Functions typically take an object as an `input`, perform an operation on that object, and then return an `output` object.

```
object <- function('input') {

    perform operation(s) on 'input'

    return output
}
# view object
object
```

Creating objects

R comes with a variety of functions for creating objects. We will start with `c()`, which stands for 'combine' or 'concatenate'.

We can print this to the console by supplying the new object and hitting enter/return.

```
x <- c(42, 34, 28, 53, 71, 30, 23, 72, 59, 46, 64, 33, 42, 50, 68)
x
```

```
##  [1] 42 34 28 53 71 30 23 72 59 46 64 33 42 50 68
```

A quick note on printing: notice the preceding `[1]` in the output. This is not part of the object, it's the line number for the output.

Now that we have an object in R, what do we do with it? We will start by taking a look at some of it's technical information using `class()` and `str()`

```
class(x)
```

```
## [1] "numeric"
```

The `class()` function tells us `x` is a `numeric` vector. The `str()` function is an abbreviation for 'structure', and it gives us a bit more information.

```
str(x)
```

```
##  num [1:15] 42 34 28 53 71 30 23 72 59 46 ...
```

I recommend using `str()` and `class()` when you're programming in R. Knowing what kind of object you're dealing with will help you determine what you can do with it.

Store and explore

Given the relationship between functions and objects, a common workflow is '*store and explore*, where we create (or import) some data as an object, apply a function to this object, store the output from this function in a new object, and use *another* function to view the result.

Store

Below we create `my_data`, a vector of 12 numbers, and print it to the console.

```
# create data
my_data <- c(49, 147, 74, 90, 7, -79, 190, 49, -123, -325, 143, 232)
my_data
```

```
##  [1]   49  147   74   90    7  -79  190   49 -123 -325  143  232
```

Next, we apply the `sqrt()` function to this object, and store the output from this function in a new object called `my_result`.

```r
# apply a function and store in my_result
my_result <- sqrt(my_data)
```

```
## Warning in sqrt(my_data): NaNs produced
```

Warnings and messages

R usually tells us when we've used a function and it's produced a result we might not expect (like missing values). In this case, this missing values aren't important, but it's a good idea to pay attention to any warnings or messages that are printed to the console.

Explore

Now we use the `summary()` function to explore the contents of `my_result`:

```r
# explore the result
summary(my_result)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   2.646   7.000   9.487   9.759  12.124  15.232       3
```

This process (applying functions to data objects, storing the results, and using functions to view their contents) makes up the majority of the R workflow.

Operators

Operators are symbols (or collections of symbols) for performing arithmetic (+, -, *, /), comparisons (<, >, =<, =>), and assignment (<- and =). These aren't a new kind of object, though (*operators* are also functions! See below).

```r
class(`+`)
```

```
## [1] "function"
```

```r
class(`<=`)
```

```
## [1] "function"
```

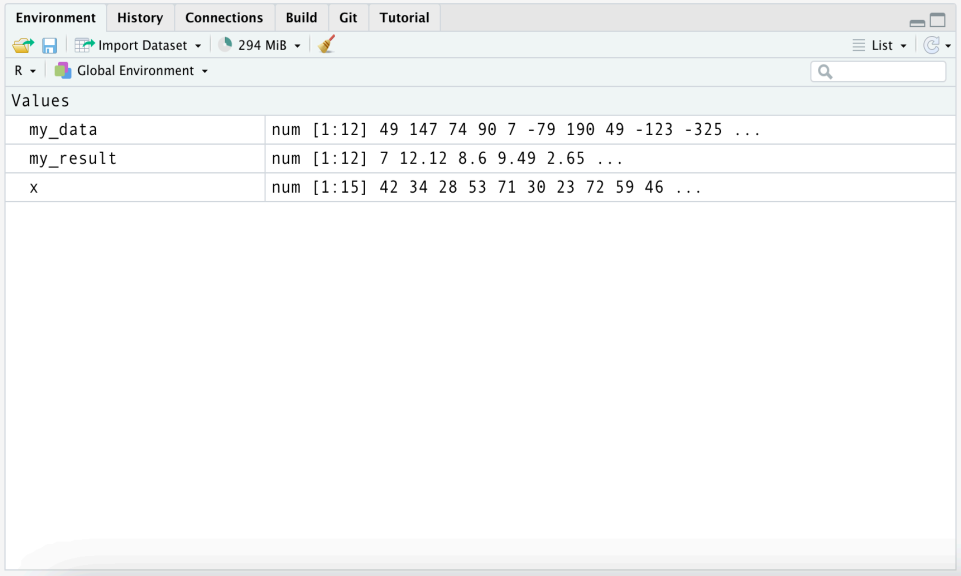
```r
class(`<-`)
```

```
## [1] "function"
```

Data in R

Before we can do anything to a particular source of data (manipulate, analyze, visualize, model, etc.), we need to import in into the RStudio environment.

Right now we have three objects in our R environment (`my_data`, `my_result`, and `x`)



```
Values
  my_data      num [1:12] 49 147 74 90 7 -79 190 49 -123 -325 ...
  my_result    num [1:12] 7 12.12 8.6 9.49 2.65 ...
  x            num [1:15] 42 34 28 53 71 30 23 72 59 46 ...
```

If we want to remove an object from the environment, we can use `rm()`.

```
rm(x)
```

To remove more than one object, separate them with commas.

```
rm(my_result, my_data)
```

Now we have an empty environment.

Loading data from packages

R comes 'out of the box' with a lot of data in the `datasets` package, which we can view using the command below:

```
library(help = "datasets")
```

To see the entire list of data–from datasets and any other R packages you've installed–use the `data()` function.

```
data()
```

Importing data

If we want to import a data from a local folder (like in our `Downloads` folder), we can use an import function like those from the `readr` package.

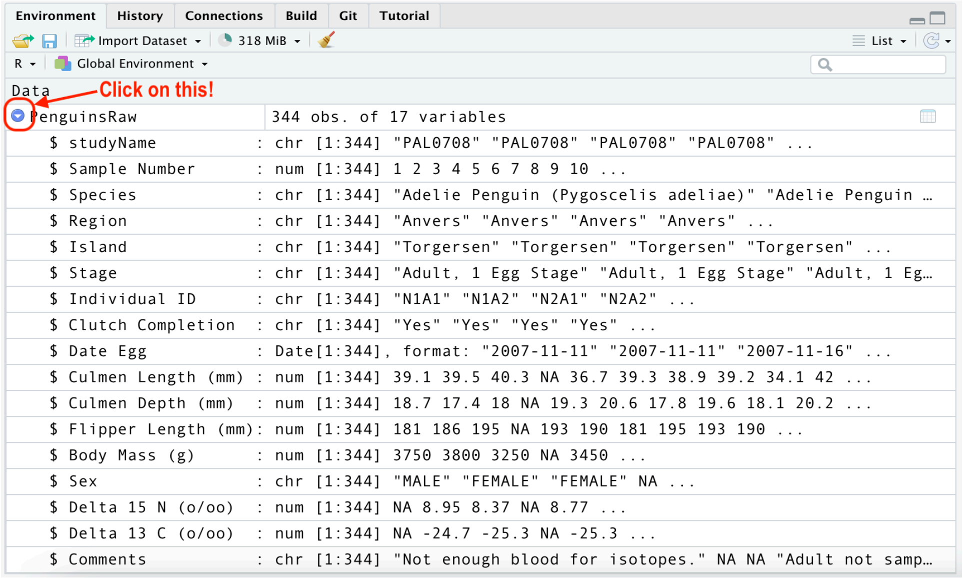
```
library(readr)
PenguinsRaw <- readr::read_csv(file = "~/Downloads/palmerpenguins.csv")
```

```
## Rows: 344 Columns: 17
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (9): studyName, Species, Region, Island, Stage, Individual ID, Clutch C...
## dbl  (7): Sample Number, Culmen Length (mm), Culmen Depth (mm), Flipper Leng...
## date (1): Date Egg
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can see the `read_csv()` printed a lot of messages to the console to let us know a bit about how the data were imported. We can also check the structure of our data by clicking on it in the **Environments** pane.



Click on this!

```
Data
PenguinsRaw                 344 obs. of 17 variables
  $ studyName          : chr [1:344] "PAL0708" "PAL0708" "PAL0708" "PAL0708" ...
  $ Sample Number      : num [1:344] 1 2 3 4 5 6 7 8 9 10 ...
  $ Species            : chr [1:344] "Adelie Penguin (Pygoscelis adeliae)" "Adelie Penguin …
  $ Region             : chr [1:344] "Anvers" "Anvers" "Anvers" "Anvers" ...
  $ Island             : chr [1:344] "Torgersen" "Torgersen" "Torgersen" "Torgersen" ...
  $ Stage              : chr [1:344] "Adult, 1 Egg Stage" "Adult, 1 Egg Stage" "Adult, 1 Eg…
  $ Individual ID      : chr [1:344] "N1A1" "N1A2" "N2A1" "N2A2" ...
  $ Clutch Completion  : chr [1:344] "Yes" "Yes" "Yes" "Yes" ...
  $ Date Egg           : Date[1:344], format: "2007-11-11" "2007-11-11" "2007-11-16" ...
  $ Culmen Length (mm) : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
  $ Culmen Depth (mm)  : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
  $ Flipper Length (mm): num [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
  $ Body Mass (g)      : num [1:344] 3750 3800 3250 NA 3450 ...
  $ Sex                : chr [1:344] "MALE" "FEMALE" "FEMALE" NA ...
  $ Delta 15 N (o/oo)  : num [1:344] NA 8.95 8.37 NA 8.77 ...
  $ Delta 13 C (o/oo)  : num [1:344] NA -24.7 -25.3 NA -25.3 ...
  $ Comments           : chr [1:344] "Not enough blood for isotopes." NA NA "Adult not samp…
```

Other tricks for getting data